SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

Method and Apparatus for Packet Analysis in a Network

Copyright Statement

Background of the Invention

[0001] The present invention relates generally to communication networks and, more particularly, to monitoring communication networks.

[0002] The providers and maintainers of data network services need to be able to collect detailed statistics about the performance of the network. These statistics are used to detect and debug performance problems, provide performance information to customers, help trace network intrusions, determine network policy, and so on. A number of network tools have been developed to perform this task. For example, one approach is to use a "packet sniffer" program such as "tcpdump" that extracts packets from the network, formats them, and passes them to a user-level program for analysis. While this approach is very flexible, it is also very slow -- requiring extensive processing for each packet and numerous costly memory transfers. Moreover, moderately priced hardware, such as off-the-shelf personal computer hardware, cannot keep pace with the needs of high-speed networks, for example such as the emerging Gigabit Ethernet standard.

[0003] Another approach is to load a special-purpose program into the network

interface card (NIC) of a network monitoring device. Processing such as filtering, transformation and aggregation (FTA) of network traffic information can be performed inside the NIC. This approach is fast -- but inflexible. As typically implemented in the prior art, the programs are hard-wired to perform specific types of processing and are difficult to change. Network operators typically require a very long lead time as well as interaction with the NIC manufacturer in order to change the program to perform a new type of network analysis.

Summary of the Invention

[0004] A method and system for monitoring traffic in a data communication network and for extracting useful statistics and information is disclosed. In accordance with an embodiment of the invention, a network interface card has a run-time system and one or more processing blocks executing on the network interface. The run-time system module feeds information derived from a network packet to the processing modules which process the information and generate output such as condensed statistics about the packets traveling through the network. The run-time system module manages the processing modules and passes the output to a host. The run-time system and the processing modules interact using a small well-defined application program interface provided for that purpose. The network monitor can be configured with the run-time system and an arbitrary collection of processing blocks, which use the application program interface and which fit into memory and timing constraints. In accordance with an aspect of the invention, the processing performed by the processing modules can be specified in a high-level language that is readily translated into a form used with the run-time system to create a new executable that is loaded into the network interface card. The processing modules can be instantiated as processing templates that are selected for execution and passed parameters without a need for generating a new executable. Alternatively, the run-time system can be enhanced to include facilities for loading and dynamically linking new processing modules on-the-fly. The processing modules thereby can be readily removed, changed, and/or replaced without replacing the run-time system module.

[0005] The present invention thereby permits a network monitor to be easily modified

as needed to manage the network infrastructure. These and other advantages of the invention will be apparent to those of ordinary skill in the art by reference to the following detailed description and the accompanying drawings.

Brief Description of the Drawings

[0006] FIG. 1 shows a block diagram of a network monitoring system illustrating various aspects of the present invention.

[0007] FIG. 2 sets forth a flowchart of processing performed by a run-time system and FTA blocks, illustrating a preferred embodiment of an aspect of the invention.

[0008] FIG. 3, 6, and 9 set forth test queries for processing a packet, specified in a high level language.

[0009] FIG. 4, 7, and 10 set forth flowcharts of processing performed by an FTA block, illustrating an embodiment of an aspect of the invention, and corresponding to the test queries set forth in FIG. 3, 6, and 9 respectively.

[0010] FIG. 5, 8, and 11 set forth the test queries set forth in FIG. 3, 6, and 9 respectively as translated into a low-level language for processing a packet.

[0011] FIG. 11 and 12 set forth test queries for processing a packet, specified in a high level language.

[0012] FIG. 13 sets forth the test query set forth in FIG. 12 as translated into a low-level language for processing a packet.

Detailed Description of the Invention

[0013] FIG. 1 shows a block diagram of a network monitoring system illustrating various aspects of the present invention. A network interface card (NIC) 110 provides an interface between a data communication network and a host computer 150. The NIC 110 interfaces to the data network at 100. The NIC 110, as is well known in the art, can comprise one or more on-board processors, hardware interfaces to the appropriate network and host, and memory which can be used to buffer data received from the data network and for storing program instructions for off-loading data processing tasks from the host. For example, and without

limitation, the NIC 110 can be a programmable Gigabit Ethernet PCI local bus adaptor manufactured by vendors such as 3Com Corporation. The host computer 150 as is well known in the art can include any device or machine capable of accepting data, applying prescribed processes to the data, and supplying the results of the processes: for example and without limitation a digital personal computer having an appropriate interface for the NIC, e.g. a PCI local bus slot. The present invention is not limited to any particular host or NIC architecture; nor is it limited to any particular communication network protocol such as Ethernet.

[0014] The software design of the NIC is represented abstractly in FIG. 1 as separate modules 120, 131, 132, 133, in accordance with a preferred embodiment of an aspect of the invention. The run-time system 120 is a flexible program that is loaded into the memory of the NIC 110 and executed on the on-board processor. The run-time system 120, in addition to performing tasks such as scheduling and resource management, handles interactions between the network 100, the host computer 150, and one or more processing blocks referred to by the inventors as "FTA processing blocks" or "FTA blocks", depicted in FIG. 1 as 131, 132, and 133. The run-time system 120 provides an environment for the execution of the FTA blocks 131...133. The FTA blocks 131...133 are program instructions that are loaded into the memory of the NIC 110 and executed on the same or a different on-board processor ("FTA" stands for filter, transform, and aggregate, although the processing capable of being performed by an FTA block is not so limited). FTA blocks 131...133, as further described herein, are preferably written in a higher-level language and compiled for use with the run-time system 120. Data such as packets traveling through the communication network are captured by the run-time system 120 and fed to the FTA blocks 131...133 residing on the NIC 110. The FTA blocks 131...133 process the packets and create output which is, in one embodiment, a condensed representation of the packets provided to the FTA. The output is gathered by the run-time system 120 and relayed to the host 150.

[0015] For example, FIG. 2 sets forth a flowchart of processing performed by the run-time system 120 and the FTA blocs 131...133. At step 201, the run-time system 120 allocates and initializes the various instances of the FTA blocks 131...133. It is advantageous to provide dynamic memory allocation for the FTA blocks, for

example so that they can store state, as well as provide for dynamic installation of new FTA blocks. At step 202, the run-time system 120 receives a packet and performs some initial processing of the packet, such as parsing the fields of the packet, perhaps doing some basic translation, and storing the fields in an appropriate data schema. At step 203, the run-time system 120 begins notifying the FTA blocks 131...133 that a packet has arrived and is ready for processing. An ordered list of FTA blocks can be utilized to determine the processing order. It is advantageous to also prioritize the FTA processing with priority levels so that FTA blocks with a lower priority level than the current resource priority level, as ascertained by the run-time system 120, can have its resources removed and reallocated once more resources are available.

[0016] The FTA blocks 131...133, in turn, receive notification from the run-time system 120 and perform processing in accordance with steps 204, 205, and 206 in FIG. 2. At step 204, the first FTA block in the ordered list retrieves only the fields in in the packet that it needs for processing. At step 205, the FTA block then proceeds to process the information in the selected fields, for example by testing predicates and recalculating aggregates. Within resource constraints, the FTA block can perform arbitrary computations. At step 206, the FTA block optionally stores the results of the processing in a named data buffer, which the inventors refer to herein as a data "tuple." It is usually advantageous for the FTA block to process additional input tuples before producing an output tuple. The FTA block then informs the run-time system 120 that it has completed its processing by returning a value. At step 207, the run-time system 120 notifies the next FTA block to commence processing, and so on, until all of the FTA blocks of the appropriate priority level have completed their respective processing tasks (it may also be useful to provide a "hook" enabling an FTA block to "short-circuit" the remaining processing on the packet, e.g. by returning a special value to the run-time system 120).

[0017] At step 208, the run-time system 120 can transfer the data tuples up to the host 150, for example on a regular basis or when the tuple buffers have filled up. Notably, the run-time system need not know anything about the format of the data, which can be private between the particular FTA block that generates the

data and the host. The run-time system 120 can also generate a system-related data tuple to inform the host 150 of system events, such as a priority-based suspension of an FTA block. At step 209, the run-time system can receive and respond to commands from the host 150, for example by removing FTA blocks, installing new FTA blocks, passing parameters to existing FTA blocks, etc.

[0018] It is advantageous for the FTA blocks 131...133, as well as the host 150, to interact with the run-time system 120 using a small application program interface (API). The following is an example of such an API. The run time system interacts with the FTA block by calling FTA block procedures, e.g. procedures called at FTA block construction and destruction time and also at other exceptional events such as flush requests, monitoring requests, notification of new operating parameters, and so on. An FTA processing block, with associated functions, can be defined as follows in the C programming language:

```
struct FTA {
        unsigned stream_id; /* id of stream contributes to */
        unsigned priority;  /* priority of FTA */

        /*  allocate and initialize parameterized instance of FTA */
        struct FTA * (*alloc_fta) (unsigned stream_id, unsigned priority,
                int argvc, void *argv[]);

        /*  release all resources associated with FTA */
        int (*free_fta) (struct FTA *);

        /*  send control data from host to intance of FTA */
        int (*control_fta) (struct FTA *, int argvc, void *argv[]);

        /*  process packet received on interface */
        int (*accept_packet) (struct FTA *, struct packet *)
}
```

The "accept_packet" function notifies the FTA block that a new packet is available for

processing and, thus, includes programming instructions for the FTA block's processing as defined by the particular user.

[0019] The run-time system, on the other hand, provides access to various procedures that an FTA block can call. It is advantageous to provide general utility functions for the FTA blocks such as functions for string manipulation, prefix matching, memory allocation, etc., so that the only library functions that an FTA block needs to call are already included in the run-time system. Functions can be included to accessing the properties of a packet and for outputting data to the host. For example, the following functions can be defined to permit an FTA to allocate memory for a data tuple and to deliver a data tuple to the host:

```
void * allocate_tuple(struct FTA *, int stream_id, int size);
int post_tuple(void *tuple);
```

Before a tuple can be output, a tuple memory block is allocated in the tuple memory space using the "allocate_tuple" function. The FTA block calls the "post_tuple" procedure to request delivery of the data to the host. It is useful to provide functions for dynamic memory allocation for FTA blocks, e.g. so that they can store state, etc.:

```
void *fta_alloc(struct FTA *, int size);
void fta_free(struct FTA *, void *mem);
void fta_free_all(struct FTA *);
```

Note that it is advantageous for the allocator to keep track of which FTA owns which block so that all allocated blocks for a given FTA can be freed all at once.

[0020] Finally, the run-time system responds to commands from the host, e.g. such as functions for receiving data that correspond to the above functions for posting data to the host:

```
int tuple_open(int stream_id);
```

```
int tuple_get_buffer(int handle, void *tuple_buffer, int buffer_size);
int tuple_close(int handle);
int tuple_reset_buffer();
```

The "tuple_get_buffer" functions retrieves the information from the network interface card. The "tuple_reset_buffer" function can be used to flush all pending data tuples. Functions ca be provided for installing and removing FTA blocks dynamically:

```
int fta_insert(int adaptor, struct FTA * after, struct FTA * new);
int fta_remove(int adaptor, struct FTA * id);
```

The following call advantageously can be used to create a new parameterized instance of a previously installed FTA block. The FTA template IDs are defined in an include file, and the function call results in a call to the FTA block's "alloc_fta" function:

```
struct FTA *fta_alloc_instance(int adapter, unsigned FTA_templat_ID,
        unsigned stream_id, unsigned priority, int argvc, void * argv[]);
```

Callouts to a corresponding FTA block can be made by a host by the following functions:

```
int fta_free_instance(int adapter, struct FTA *FTA_id);
int fta_control(int adapter, struct FTA *FTA_id, int argvc, void *argvc[])
```

It is also advantageous to define a scratchpad and associated functions in situations where long parameters need to be passed to the FTA blocks.

[0021] As noted above, the processing performed by an FTA block can be arbitrary. Although the processing of an FTA block can be specified in a low-level programming language like C using the above-specified API directly, it is more advantageous to cast the problem of writing an FTA block as one of writing a query on a relational table where the table corresponds to the stream of packets in the

communication network. The query can be written in a language such as SQL or a subset of SQL and automatically transformed or translated into code usable to instantiate the FTA block. This reduces the creation time and allows non-specialists to write FTA blocks. FIG. 3, 6, and 9 set forth various test queries describing packet processing in a high-level representation.

[0022] FIG. 3 sets forth an example query that is used to select and record information on packets that are larger than a specified size. At lines 305-306, the query specifies that the "timestamp" and "hdr_length" field should be retrieved from an IPv4 packet for processing. At line 307, a test predicate is defined for the query which chooses packets which have a "hdr_length" greater than 50. This query can be parsed and utilized to generate processor instructions in a programming language such as the C programming language taking advantage of the API specified above, as set forth in FIG. 5A and 5B. FIG. 4 sets forth a simplified flowchart of the processing that would be performed by the FTA block with this simple query. Starting at line 528 in FIG. 5A, the "accept_packet" function is defined which specifies the processing to be performed by the FTA block. At lines 539-543, the FTA block retrieves the fields referenced above from the packet using standardized functions for retrieving packet information. This corresponds to step 401 shown in FIG. 4. Then, at lines 546-548, the FTA block determines whether the packet meets the defined test predicate, i.e. whether "hdr_length" > 50. This corresponds to step 402 shown in FIG. 4. If the packet does not meet the test predicate, it is ignored. If the packet does meet the test predicate, a tuple is created and posted at lines 550-558, corresponding to step 403 in FIG. 4. The data structure of the tuple stores information corresponding to the fields selected in the query.

[0023] FIG. 6 sets forth another example query that is used to count packets. As set forth in lines 606-608, a "count" of packets is maintained for every 5000 time units, as recorded in the timestamp. The 5000 time units are grouped and defined at line 608 as a "timebucket." The query records the timebucket and the count of packets for the timebucket. The query set forth in FIG. 6 readily translates into the flowchart set forth in FIG. 7 and the programming code set forth in FIG. 8A, 8B, and 8C. FIG. 7 describes the processing performed by the FTA block,

corresponding to the "accept_packet" function defined starting at line 814 in FIG. 8B. At step 701, the referenced packet fields are unpacked, corresponding to line 833-834 in FIG. 8B. The only information about the packet that the FTA block needs to satisfy this query is the timestamp of the packet. There is no test predicate in the query, so the FTA block may then proceed with determining how to process the timestamp information. At step 702, the FTA block determines whether the temporal attribute has changed, namely whether the packet belongs to the current timebucket or a new timebucket (this corresponds to lines 839-844 in FIG. 8B). If the packet belongs to a new timebucket, the FTA block at step 703 proceeds to flush the current aggregate count and create an output tuple(s) for the aggregates that are being kicked out. The FTA block no longer needs this aggregate count since no future packets received can contribute to a past timebucket. Step 703 corresponds to the function "fta_aggr_flush" function defined at lines 838-61 in FIG. 8A. An output tuple is created for the aggregate and posted at lines 847-854. The "fta_free" function, mentioned above, is then utilized at line 856 to flush the current aggregates to make room for the new timebucket. As described above, the run-time system can keep track of which FTA is using which memory and can, in an embodiment, advantageously perform garbage-collecting.

[0024] With reference again to FIG. 7, the aggregate count is incremented with the arrival of the new packet and an output tuple produced if aggregate storage space must be reclaimed at steps 704 to step 709. This corresponds to lines 801 to 844 in FIG. 8C. The complexity of the processing performed at this part of the FTA block is useful as more complex type of aggregates are expressed in the query.

[0025] FIG. 9 sets forth a more complicated example query that counts packets that satisfy a certain test predicate. The test predicate is set forth in lines 908 to 910 in FIG. 9. The information to be retrieved from the IPv4 packet, again, are set forth in the "select" statement at lines 905 and 906 in the query. The data is to be aggregated by timestamp and "hdr_length" as specified in line 911. This query readily translates into the flowchart set forth in FIG. 10 and the programming code set forth in FIG. 11A, 11B, 11C, and 11D. FIG. 10 again describes the processing performed by the FTA block, corresponding to the "accept_packet" function defined starting at line 1117 in FIG. 11B. At step 1001, the referenced packet fields

are unpacked, corresponding to lines 1138-1148 in FIG. 11B. At step 1002, the FTA block performs packet filtering by determining whether the test predicate is satisfied, corresponding to lines 1150-1156 in FIG. 11B. As specified in the query, packets are chosen based on the TTL field and the timestamp. If the test predicate is not satisfied, then it is ignored. If it is satisfied, the information retrieved from the packet is utilized in the computation of the aggregates at steps 1003-1008 in FIG. 10. At step 1003, the FTA block searches for an aggregate that matches on the group by attributes. This corresponds to lines 1101-1112 in FIG. 11C. If a match is found, at step 1004, then the aggregate is updated in place and moved to the front. This corresponds to lines 1115-1126 in FIG. 11C. If a match is not found, and there is room in the aggregate list, at step 1006, then another aggregate block is added to the aggregate list, at step 1008 in FIG. 10. This corresponds to lines 1154-1160 in FIG. 11C. If there is no room in the aggregate list, then, at step 1007, space is reclaimed from the end of the list and an output tuple is created for the aggregate being kicked out. This corresponds to lines 1130-1151 in FIG. 11C. Thus, the FTA block can compute the aggregates specified in the query, even given the limited memory resources of a typical network interface card.

[0026] FIG. 12 sets forth an example query that illustrates how the system can allow the queries to make calls to certain functions defined in the run-time system. The function "str_find_substr" finds substrings and can be included in the run-time system. An include file can be defined that contains the prototypes of all functions that a query can access. The information to be retrieved from the IPv4 packet are set forth in the "select" statement at lines 1205-1206 in the query. The test predicate is set forth in lines 1208-1209, which utilizes the "str_find_substr" function. If the indicated substring, "host:*\n", cannot be found, it is the equivalent of an attribute that cannot be retrieved, i.e. the query discards the tuple. The query, at lines 1210, also aggregates the results by "hdr_length" as further described above.

[0027] FIG. 13 sets forth another example query that illustrates how parameters may be passed to a FTA block. FIG. 13 sets forth a simple query that accepts parameters. FIG. 14A and 14B set forth the generated C code that corresponds to

the query in FIG. 13. The parameter, as set forth in line 1303 in FIG. 13, is "min_hdr_length" which is defined in the "DEFINE" statement as an integer value. The DEFINE statement may be used to define the type of the parameter. In the query, at line 1308, the parameter is referenced in the test predicate by "$min_hdr_length". In the generated programming code, the parameter reference is translated into a reference set forth as "param_min_hdr_length" at line 1402 in FIG. 14B. The "load_params" function, defined at lines 1419-1428 in FIG. 14A, accepts new parameter values for "param_min_hdr_length" and whatever other parameters are defined in the query. Thus, the same FTA block advantageously may be utilized and reused, with different parameters passed to the FTA block.

[0028] The foregoing Detailed Description is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from the Detailed Description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention.